Extracted from:

Genetic Algorithms in Elixir

Solve Problems Using Evolution

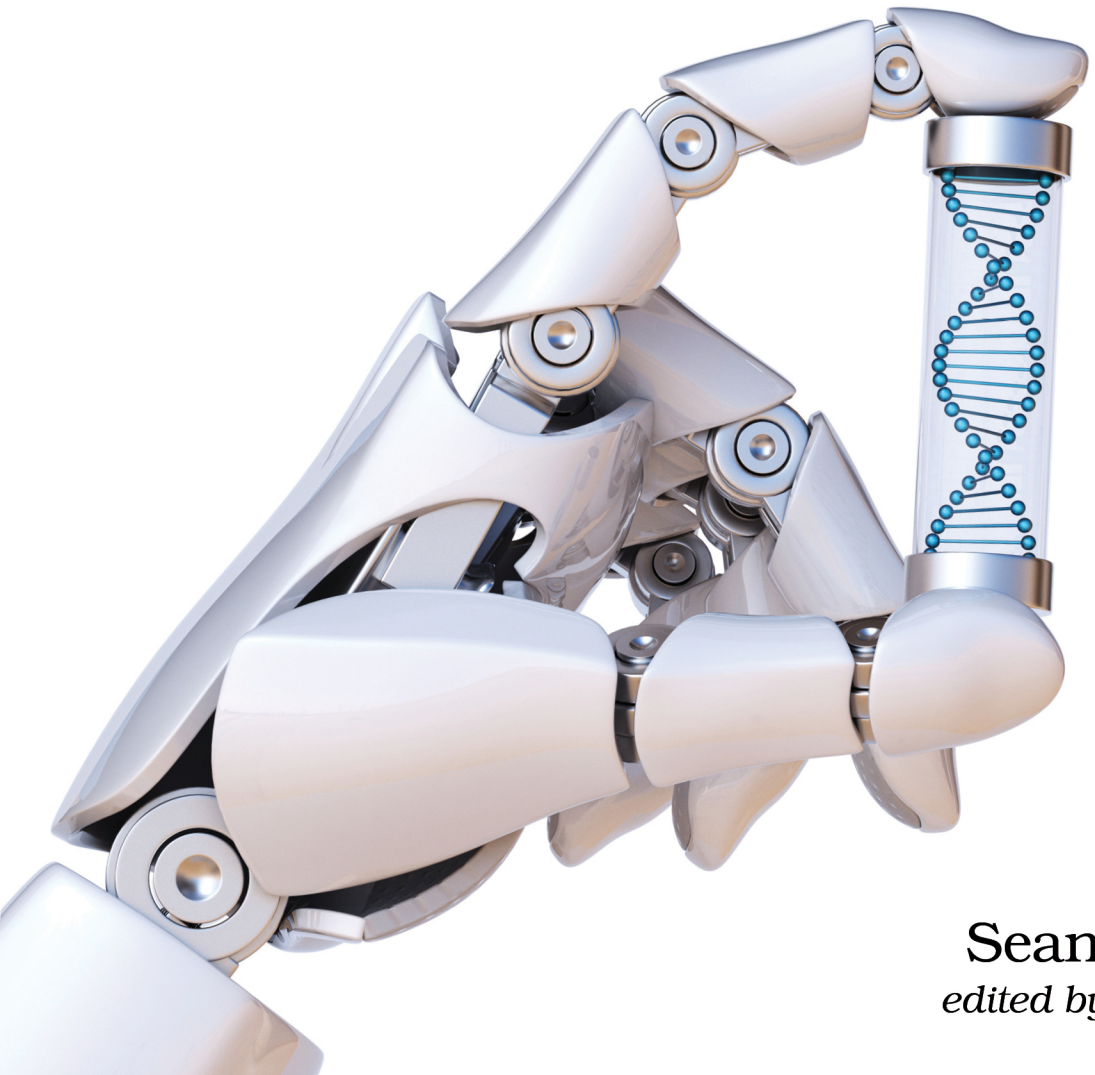
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Genetic Algorithms in Elixir

Solve Problems Using Evolution



Sean Moriarity

edited by Tammy Coron

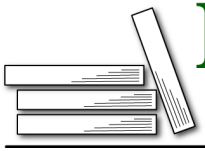
Genetic Algorithms in Elixir

Solve Problems Using Evolution

Sean Moriarity

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Development Editor: Tammy Coron
Copy Editor: L. Sakhi MacMillan
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-794-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—January 2021

In a world of competition, people are always searching for the best. The best job, the best diet, the best financial plan, and so on. Unfortunately, with so many options, it's impossible to make the best decisions all the time. Fortunately, humans have evolved to navigate the complexity of everyday life and make informed decisions that ultimately lead to success.

While your brain is naturally wired to make informed decisions, computers are not. Computers are naive—they can only do what you program them to do. So how do you program a computer to make informed decisions, and why is this even necessary?

Consider this example: you're tasked with designing the shipping route for a large shipping company. You're given a list of fifteen cities and your job is to pick the shortest route between them to save the company money on gas and travel expenses. At first, you might think it's best to calculate every possible path between the cities—there're only fifteen. Unfortunately, the number of possible paths is 130,766,744,000,000—that's 130 *trillion*. This problem is an example of the *traveling salesman problem*. The goal of the traveling salesman problem is to find the shortest route between a designated number of cities.

The number of possible paths grows at a *factorial* rate. A factorial is the product of every integer up to a certain integer. In the shipping example with fifteen cities, you can calculate the number of paths by multiplying every integer from 1 to 15.

Nobody has enough time to calculate the distance of 130 trillion paths. You have to take a better, more informed approach. You could choose a random start point and choose to travel to the next closest city after every stop. This strategy might produce the shortest path—you could even calculate the distance of the paths produced from starting at every city and choose the shortest one from that. You'd then only have to calculate the distance of fifteen paths. Unfortunately, experimenting with different strategies is still time consuming, and without a calculated approach you might miss the best strategy.

So how can you make the best decisions and teach a computer to do the same?

The answer is *optimization*. Optimization is the practice of making the best possible decisions in a situation. You can think of optimization as the search for the best. Humans are great at optimizing—it's natural for us to find and make the best decisions for ourselves. Computers can be great at optimizing too; they just need a little help.

Optimization *algorithms* are techniques for solving optimization problems—problems where your goal is to find the best of something. An algorithm is a series of instructions. An optimization algorithm is a set of instructions for finding the best solution to a problem. While there are countless optimization algorithms, one of the oldest and most common is the *genetic algorithm*.

Understanding Genetic Algorithms

Genetic algorithms are a class of optimization algorithms based on evolution and natural selection. They use strategies loosely based on genetics and biology to produce optimal—think "best"—or near-optimal solutions to complicated problems. Initially conceived in the 1960s, the intended use for genetic algorithms was simply a technique for creating adaptable programs. Today, genetic algorithms are used in numerous applications in fields like artificial intelligence and finance. They're great at solving difficult optimization problems and lend themselves nicely to parallel computing and distributed architectures. They can even yield solutions to the shipping problem mentioned earlier.

The First Genetic Algorithm

The first genetic algorithm was introduced by John Holland at the University of Michigan in the 1960s; however, evolutionary algorithms had been around long before that. Early artificial intelligence researchers believed evolution was the key to creating truly intelligent programs. Today, the field of evolutionary computation has many, somewhat loosely defined, branches of research, such as evolution strategies, genetic programming, and genetic algorithms.

At their core, optimization problems are *search problems*. Search problems require you to navigate an area, like a maze, to find an objective, like the end of the maze. Optimization problems are basically the same thing, only there are multiple possible solutions. Imagine a maze with multiple exits. Your goal is to exit the maze as quick as possible—this means your goal is to find the shortest path to any of the maze exits.

Two basic approaches are used for search problems: *brute-force search* and *informed search*. It's important to understand the difference to understand why optimization and genetic algorithms are so useful.

Understanding Informed Search

An informed search relies on a search strategy. In an informed search, you make smart decisions based on the available information. In a brute-force search, you iterate over every possible solution linearly. Brute-force searches

use no knowledge of the search area to make decisions. In a maze, a brute-force solution would try every possible path—never stopping to consider whether or not the paths are getting smaller or larger, or if the paths will even lead to an exit. Brute-force searches are naive. Eventually you'll find a solution, but it might take a long time, and it might not even be the best one.

The key to informed search and thus optimization techniques, like genetic algorithms, lies in how they balance *exploration versus exploitation*. Imagine you find yourself lost in the woods without a map or compass. How would you navigate out of the woods?

Using Crossover to Exploit

One option is to use a brute-force strategy—walk in circles around every tree, hoping you make it back to civilization before you get too tired. Of course, if the woods are large, the brute-force strategy becomes especially difficult. Another option is to use the information around you. With this strategy, you *exploit* or take advantage of the information available to you to determine which direction to head next. To exploit in search means to use what you already know to navigate. In this example, perhaps you know that the nearest town is north, and you can tell where north is because of the position of the sun. This, in essence, is what genetic algorithms do. They use the data around them to make correct decisions.

Crossover is how genetic algorithms exploit in search. Crossover is the process of creating new *child* solutions from *parent* solutions. The idea is that the strongest solutions have characteristics that make them strong. These characteristics are called *schemas*. Schemas are building blocks of fit solutions—you'll learn more about them in Chapter 4, Evaluating Solutions and Populations, on page ?.

The term crossover is a loose analogy to genetic reproduction. While the analogy is weak and crossover in genetic algorithms isn't remotely the same as crossover in biology, it can better help you understand what's going on under the hood. Crossover is a part of how genetic algorithms make good decisions. In the woods example, you choose where to go next based on your current position. Your next step is a product of where you were last. The idea is to build progressively better solutions over time, until you reach your goal.

Using Mutation to Explore

Now, imagine that some of the information available to you is misleading. Perhaps somebody tells you there's a road that leads to the nearest town, but the road just takes you in circles around the woods. Would you continue to

repeatedly follow the road, never realizing that the path you're on isn't correct? No, you'd *explore* other paths in the woods, hoping that one would eventually lead you out. To explore in search is to try new, random paths to see if they produce a better outcome. This concept of getting stuck in the same place in the search space is parallel to a common pitfall in optimization problems known as *premature convergence*. It's easy for genetic algorithms to get stuck in one part of a search space because some solutions *appear* to be good enough—even though better solutions exist. You'll learn more about premature convergence in Chapter 7, Preventing Premature Convergence, on page ?.

Mutation is how genetic algorithms explore. It's not enough to simply keep trying to build new solutions from previous ones, which is essentially the same as trying the same path over and over again. Mutation introduces randomness into your genetic algorithms. The goal is to slightly alter some aspect of the previous solutions to create newer solutions, which may lead to newer, better paths.

The effectiveness of genetic algorithms largely relies on how you balance exploitation versus exploration. Favoring one over the other has merits. Oftentimes, if you don't know much about a search space, it's best to favor exploration first and then slowly shift toward exploiting the information you already know. This is similar to how you might learn to navigate a new town—try new things until you have enough information to take the best routes.

The best way to understand how genetic algorithms work is to create one. In this next section, you'll learn the basics of genetic algorithms by solving a very simple problem known as the *One-Max problem*.